Rev
1.0.1

ASI Bootloader Sequence Description

AUTHORED BY
EVIN BALLANTYNE

ACCELERATED SYSTEMS INC | 60 Northland Road, Unit 6, Waterloo ON

ASI ACCELERATED SYSTEMS INC.

Contents

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

Overview

The following document contains a high-level description for boot loading software to ASI devices including (but not limited to) BACs, VCMs, and throttles over UART and CAN.

What is Not in this Document

- How switch the device into boot mode. This is different for each device/firmware and can be found in the documentation there.
- Not all scenarios are covered with regards to errors. 99% of the time, the recommended solution is to resend the message on a response timeout, double check that the sequence was correctly followed or power cycle the device and try again.

References

- https://en.wikipedia.org/wiki/XML
- https://en.wikipedia.org/wiki/Base64
- https://en.wikipedia.org/wiki/Intel_HEX
- https://en.wikipedia.org/wiki/XTEA
- https://en.wikipedia.org/wiki/CANopen

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

Intel Extended Hex File Layout

The .EHX file will need to be decoded and parsed. EHX files can be viewed in any text viewer, as it's format is that of an XML file. More on XML can be found here: https://en.wikipedia.org/wiki/XML.

XML Schema

The layout of the XML text, or schema, is as follows:

doctype : string
protocol : integer
protected : boolean
checksum : hexadecimal
data : string

For the purposes of boot loading, we can ignore everything except for the **data** tag (although you can use the checksum to verify data integrity if you like). This is just a string encoded with Base64. More on Base64 can be found here: https://en.wikipedia.org/wiki/Base64. Once decoded, you will get another XML string. While this XML string has many tags, for boot loading we only need the following:

ehx-data : grouping

> **target** : string
> **sector-mask** : hexadecimal
> **hex** : grouping
>
> > **L0** : string
> > **L1** : string
> > **L2** : string
> > **L3** : string
> > **L4** : string
> >
> > …*etc.*

All other tags can be ignored. It is recommended that before you begin the boot loading sequence, you check that the **target** string matches the desired value, but this is technically not necessary. The **sector mask** will be needed to tell the UART bootloader what sectors of flash to erase, and the list of **hex-lines** will be the encrypted data loaded onto flash. **Hex lines** are in the **Intel Hex** format. More on **Intel Hex** format found here: https://en.wikipedia.org/wiki/Intel_HEX. It is recommended you take the time to understand this format if you are developing a UART bootloader as you will need to parse each hex line.

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

UART Bootloader Sequence

The following is a high level description of the bootloader sequence over UART communications such as TTL/RS232. The communication protocol is that of a “controller”/”agent” paradigm, with the user device sending requests to the bootloader, and the bootloader responding.

Uart Settings

The bootloader uses the following UART settings:

- Baud rate = 115200 bps
- Stop bits = 1
- Parity = None
- Word length = 8 bits

Uart Message Structure

The following is the message structure for all request and response messages:

DATA	COMMAND	CRC	EOM
(0 to 36 Bytes)	*(0 or 1 Byte)*	*(0 or 2 Bytes)*	*(2 Bytes)*

Note that all byte structures within the message are Big Endian.

DATA: 0 to 36 Bytes of Data depending on the command. It is possible to have no Data.

COMMAND: Signifies the operation expected from the device on a request. For the most part, responses have the same command. It is possible to have no command provided there is also no Data.

CRC: Calculated for each byte in the message and occupies two bytes. CRC can be calculated with the below Pseudocode:

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

```
unsigned short CRC16Lookup[] =
      0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
      0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF

unsigned short getCRC16(unsigned byte bytes[] =>

      unsigned short crc = 0xFFFF

      foreach(b in bytes)
            unsigned short index = (crc >> 12) ^ (b >> 4)
            crc = (crc << 4) ^ CRC16Lookup[index]
            index = (crc >> 12) ^ (b & 0x0Fu)
            crc = (crc << 4) ^ CRC16Lookup[index]

      return crc
```

Note that there will be no CRC if there is no Data and no Command (see sequence commands).

EOM: End of Message. Will consist of a sequence command. All requests will have the "End of Message" Sequence command, while responses can have other sequence commands denoting errors such as a bad CRC.

UART Requests

Name	Data Bytes	Data Layout	Command
Echo	0	(No data)	0xD0
Get Chip ID	0	(No data)	0xD6
Get Seed	0	(No data)	0xDC
Unlock Flash	8	64-bit seed that has been decoded, inverted, and encoded	0xD4
Get Flash API Version	0	(No data)	0xD7
Erase Flash	2	16-bit Sector Erase Mask Bit 0: Erase sector A Bit 1: Erase sector B Bit 2: Erase sector C Bit 3: Erase sector D Bit 4: Erase sector E Bit 5: Erase sector F Bit 6: Erase sector G Bit 7: Erase sector H	0xD1

Program Flash	Up to 36	First 4 Bytes: Starting memory address Remaining Bytes: Program Data	0xD5
Reset	0	(No Data)	0xD8

UART Responses

Name	Data Bytes	Response Layout	Command
Echo	0	(No data)	0xD0
Get Chip ID	4	First 2 Bytes: Part ID Last 2 Bytes: Revision	0xD6
Get Seed	8	64-bit random seed	0xDC
Unlock Flash	2	0x0000 = Unlocked 0x0010 = Not Unlocked	0xD4
Get Flash API Version	2	16-bit version number	0xD7
Erase Flash	2	0x0000 = Flash Erased 0x0001 = Bad Sector Mask	0xD1
Program Flash	Up to 6	If address in request is out-of-bounds, first 2 bytes are 0x0001. Next (or first) 4 bytes are the address sent in the request. Last 2 Bytes: 0x0000 = Success 0x0001 = Data sent was too large	0xD5
Negative Acknowledge	4	First 2 Bytes: Command Sent Last 2 Bytes: 0x12 = Message unsupported 0x15 = Option unsupported 0x16 = Buffer Overflow 0x17 = Illegal Address 0x18 = Access Denied 0x19 = Bad App CRC	0x7F

Sequence Command Codes

Sequence command Codes are byte stuffed commands that while most often used to signal the end of a message, can appear ANYTIME within a UART Message. It will always begin with the byte **"0xFF"** with the following byte representing the special or notice. If "**0xFF**" is valid data instead, then the next byte will also be "**0xFF**". Take care of noting this when parsing and creating Uart messages.

Sequence Command	Meaning
0xFFF0	Remote Ping Command (unused in ASI bootloaders)
0xFFFC	Overflow Error
0xFFFD	CRC Error
0xFFFE	End of Message
0xFFFF	Data Byte of **0xFF**

Initialization Sequence

This is how you initialize stuff:

1. Send **Echo** Request
    - Expect **Echo** Response
2. Send **Get Chip ID** Request
    - Expect **Get Chip ID** Response
    - Ensure returned chip parts and revision match desired product
3. Send **Get Seed** Request
    - Expect **Get Seed** Response
    - Data will be a 64 bit random seed
4. Run Random Seed through *xteaDecode* operation
    - Algorithm can be found here: https://en.wikipedia.org/wiki/XTEA
    - Key provided by ASI depending on device and bootloader version.
5. Perform *Bit Inversion* operation on decoded random seed
    - randomSeed = ~randomSeed in most coding languages.
6. Run Random Seed through *xteaEncode* operation
    - Algorithm can be found here: https://en.wikipedia.org/wiki/XTEA
    - Key provided by ASI depending on device and bootloader version.
7. Send **Unlock Flash** Request
    - Data is Re-Encoded random seed.
    - Expect **Unlock Flash** Response
    - Ensure data from response is *UNLOCKED*
8. Send **Get Flash API Version** Request

- Expect **Get Flash API Version** Response
- Ensure returned version is desired

9. Send **Erase Flash** Request
    - Data is a bit vector of sectors to erase
    - Expect **Erase Flash** Response
    - Ensure data from response is *FLASH ERASED*
    - (Note: the Erase procedure can take several seconds and the response message comes AFTER the erase procedure takes place. An 8 second timeout is recommended on the response message).

Program Sequence

For each HEX LINE in document:

1. Convert Characters to Intel-Hex Byte Array
    - First Byte is the number of bytes in the data section
    - Second and Third Bytes are the address
    - Fourth byte is the ***RECORD TYPE***
    - The rest of the data bytes are data (if any) followed by a one-byte checksum.
    - More information on Intel-Hex Data can be found here: https://en.wikipedia.org/wiki/Intel_HEX
2. Perform the following operations:
    - If the Record Type is **Extended Linear Address**, locally store the 2 data bytes as an **UPPER ADDRESS** 16 bit value to be used later (No request sent)
    - If the Record Type is **End of File**, go to cleanup (No request sent)
    - If the Record Type is **Data**:
        - Send a **Program Flash** Request
        - First 2 Data Bytes are the **Upper Address** from earlier
        - Next 2 Data Bytes are the address of the current line
        - The rest is data from the current line

Cleanup Sequence

1. Send **Reset** Request
    - There will be no response message on a success.
    - Not all bootloaders support this function, in which case you need to instruct your users to power cycle the device.

CAN Open Bootloader Sequence

The CAN Bootloader Sequence consists of a series of CANopen SDO/SDO Expedited messages. Message structure will not be covered in this document but suffice to say a good grasp on SDO Messages will be essential. More on the CANopen protocol can be found here: https://en.wikipedia.org/wiki/CANopen.

Differences between CANopen Bootloader Sequence and UART Bootloader Sequence

- Unlike the Uart bootloader sequence, we do not choose the sectors to erase in flash. Instead, we tell the bootloader whether we will be programming the App section, or the Parameters section.
- We do not need to parse the **Intel Hex** lines; the bootloader will do that for us. We just need to pass in the characters as-is.

CAN Node ID and Baud rate

- The Node ID of the bootloader will always be **0x7E**, regardless of the programmed Node ID of the device. It is ***HIGHLY RECOMMENDED*** that you check that no other device is in boot mode before switching the device to boot mode.
- The baud rate for most ASI devices is 250kbs, regardless of the programmed baud rate of the device. Some newer devices developed by ASI will ensure the bootloader is at the same baud rate that was programmed in firmware.

CAN Object Dictionary

Index	Sub-Index	Length	Read/ Write	Name	Description
1000	00	4	RO	Device Type	
1018	01	4	RO	Vendor ID	
	02	4	RO	Product Code	
	03	4	RO	Software Version	
1F50	01	4B	RW	Program Data (App)	
	02	4B	RW	Program Data (Parameters)	

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

1F51	01	1	RW	Program Control	0x01 = Run App 0x03 = Erase Flash 0x80 = Erase Parameters 0x81 = Check Seed 0x82 = Latch Seed
1F57	01	4	RO	Flash Status	bit 0: in progress bit 1 to 7: error code *0x1 = No Program* *0x2 = Bad Data Format* *0x3 = Bad Checksum* *0x4 = Not Erased* *0x5 = Flash Write Failed* *0x6 = Bad Flash Address* *0x7 = Flash still Locked* bit 8 to 16: (unused) bit 16: Seed Latched bit 17: Unlocked bit 18: App has valid CRC bit 19: App sectors Erased bit 20: App write in progress bit 21: Parameters sectors erased bit 22: Parameters write in progress bit 23: Parameters have valid CRC bit 24 to 31: (unused)
2010	01	2	RW	Random Seed 0	unsigned short
	02	2	RW	Random Seed 1	unsigned short
	03	2	RW	Random Seed 2	unsigned short
	04	2	RW	Random Seed 3	unsigned short

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

Initialization Sequence

1. Write to **Program Control**
    - **Latch Code** Command
2. Read **Flash Status**
    - Confirm **Seed Latched** bit is high
3. Read **Random Seed 0 to 3**
    - You'll receive 4 unsigned shorts
4. Run Random Seeds through *xteaDecode* operation
    - Algorithm can be found here: https://en.wikipedia.org/wiki/XTEA
    - Note that the algorithm usually takes 2 unsigned integers (32 bits), so you will have to adjust the algorithm a little to take 4 unsigned shorts (16 bits) OR combine the 4 shorts into two integers before inserting it into the algorithm.
    - Key provided by ASI depending on device and bootloader version.
5. Perform *Bit Inversion* operation on all decoded random seeds
    - randomSeed[i] = ~randomSeed[i] in most coding languages.
6. Run Random Seeds through *xteaEncode* operation
    - Algorithm can be found here: https://en.wikipedia.org/wiki/XTEA
    - Note that the algorithm usually takes 2 unsigned integers (32 bits), so you will have to adjust the algorithm a little to take 4 unsigned shorts (16 bits) OR combine the 4 shorts into two integers before inserting it into the algorithm.
    - Key provided by ASI depending on device and bootloader version.
7. Write All Encoded Random Codes back to **Random Seed 0 to 3**
8. Write to **Program Control**
    - **Check Code** Command
9. Read **Flash Status**
    - Confirm **Unlocked** bit is high
10. Write to **Program Control**
    - For App:
        - **Erase Flash** Command
    - For Parameters:
        - **Erase Parameters** Command
11. Read **Flash Status**
    - For App:
        - Confirm **App Sectors Erased** bit is high
    - For Parameters:
        - Confirm **App Sectors Erased** bit is high
    - This operation can take a few seconds, and you will likely experience write timeouts. It is recommended you ignore timeout errors for 8 seconds.

Program Sequence

For each HEX LINE in document:

1. Write all characters in HEX LINE to **Program Data**
    - Sub Index 0x01 for App
    - Sub Index 0x02 for Parameters
    - Note that unlike the UART programming sequence, you do NOT need to parse the characters, just send the hex line as it is.
2. If HEX LINE is “End of File” Sequence, skip to “Cleanup Sequence”.
    - “End of File” looks like ":00000001FF".
    - Note that you are still sending this line to the bootloader.
3. Read **Flash Status**
    - For App:
        - Confirm **App Write in Progress** bit is high
        - Confirm **App Has Valid CRC** bit is low
    - For Param:
        - Confirm **Param Write in Progress** bit is high
        - Confirm **Param Has Valid CRC** bit is low

Cleanup Sequence

1. Wait for a minimum of 500ms
    - This is to allow time for the bootloader to calculate and burn the App CRC
2. Read **Flash Status**
    - For App:
        - Confirm **App Write in Progress** bit is low
        - Confirm **App sectors Erased** bit is low
        - Confirm **App Has Valid CRC** bit is high
    - For Param:
        - Confirm **Param Write in Progress** bit is low
        - Confirm **Param sectors Erased** bit is low
        - Confirm **Param Has Valid CRC** bit is high
3. Write to **Program Control**
    - **Run App** Command
    - Note that this write may produce a timeout error as some bootloaders simply jump to the app instead of returning a response first.

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com

ASI ACCELERATED SYSTEMS INC.

Document Revision History

Version	Author	Modifications	Date
1.0.0	Evin Ballantyne	Initial Creation	November 9, 2021
1.0.1	Evin Ballantyne	Fixing typos	May 12, 2022

60 Northland Road, Unit 6
Waterloo, Ontario, Canada
N2V 2B8

Phone: +1 (519) 342-2507
info@acceleratedsystems.com
www.acceleratedsystems.com